Victorian 6502 User Group Newsletter

KAOS

For People Who Have Got Smart

HARDWARE: . .. . DAVID ANEAR
SOFTWARE: .. .. JEFF RAE
LANGUAGES: . .. .. . TONY DURRANT
AMATEUR RADIO: ROD DRYSDALE VK3BYU
EDUCATION: .. .. NOEL DOLLMAN
LIBRARY: .. .. JANNENE
TAPE LIBRARY: .. .. RON CORK
NEWSLETTER: .. .. IAN EYLES
SYM: .. .. BRIAN CAMPBELL
ATARI: .. .. GERRY McCAUGHEY
SECRETARY: . .. .. ROSEMARY EYLES

OSI SYM KIM AIM ATARI APPLE UK101 ORANGE

Registered by Australia Post
Publication No. VBG4212

ADDRESS ALL CORRESPONDENCE TO 10 FORBES ST., ESSENDON, VIC. 3040

Vol.2 No.9 June 1982

The original concept of KAOS was to serve as an information exchange, to allow OHIO users to contact other people with similar interests, so KAOS has never been in the position to handle the sale of hardware. Hardware items mentioned in the newsletter are available from the designer/manufacturer, COMP-SOFT or LOOKY VIDEO. To assist members looking for information on hardware items, we intend to publish a directory in the July newsletter, so if you have items of hardware for sale or information about where particular items may be purchased, please supply us with the information as soon as possible. While we are on the subject, we would also like to publish a list of software that members have written and would like to sell or exchange.

TRASH and TREASURE Don't forget the sale is on at the June meeting, starting at 1pm, so bring along your money, trash or treasure. The sale will be held down stairs and members will be responsible for the safety of their goods, this will not be an auction andowners will handle the sale of their own goods. Any unsold items cannot be left at the school.

FORTH For those interested in FORTH, David Wilson would like to see you after the general meeting to discuss FORTH and forming an interest group. On the subject of FORTH, Ritchie Laird has been doing a lot of work with the cassette version and would like to hear from anyone with similar interests, see the article on page 15.

SYM members please note that Brian Campbell's new phone number is 589 2054.

At the last meeting the ATARI owners decided they had enough members to form their own group which will be known as M.A.C.E. and they will now be producing their own newsletter.

The next meeting will be held on Sunday 27th June at 2pm at the Essendon Primary School, corner of Raleigh St and Nicholson St, Essendon. The school children will be in as usual.

****

INDEX

SUPERMON - THE CASSETTE I/O ROUTINES

The cassette hardware on the SYM is just an amplifier. Unlike many other systems, there is no UART and frequency shift keying (FSK) circuit. This function is controlled by the CPU. This gives the SYM a distinct advantage in that it can read almost any cassette format as well as its own.

In the March issue of SYM-POSIUM I mentioned three subroutines for cassette I/O. These were RDBYTH, OUTBTH and SYNC. The first two routines are almost the equivalent of INCHR and OUTCHR for cassette.

Before discussing the routines, we must first see how the data is stored on tape. The method used is known as bi-phase modulation, and is the same as that used on disks. The level on the output port is inverted at the beginning of a bit. If the bit to be saved is a zero, the level remains the same for 700 micro-seconds. If the bit to be output is a one, the level is inverted again after 250 micro-seconds (SUPERMON V1.1) or 350 micro-seconds (SUPERMON V1.0). This sequence is repeated for all 8 data bits.
For example:-

This represents the number $2B since the data is sent with the least significant bit first. Since this system works on the changes in level and not the level itself, the above signal can be inverted without altering the result. You will also notice that there is no need for the usual start and stop bits as is necessary for many other systems, although there is a delay between bytes to allow the system to store the data.

If a block of data was just sent straight to tape, the SYM would have extreme difficulty retrieving it. Firstly, it would not know where it is in the file (ie. before the start, in the middle, etc.), whether it is the right file in the first place and where to put all the incoming data. For this reason the file is formatted in a particular way.

At the start is a series of sync characters. This is just a string of $16 stored on a few seconds of tape. This allows the SYM to get into synchronism with the incoming data stream. An "*" (hex $2A) indicates the end of the sync pattern and prepares the SYM for the main part of the file.

The first byte to come off tape is the ID number. If this is correct the rest of the file will be loaded, otherwise the system will resume its search for sync. The ID number is displayed on the left hand seven segment display. The 'a' segment corresponds to bit 0, 'b' to bit 1, and bit 7 lighting the decimal point.

The next four bytes give the start address and the end address + 1. This tells the SYM where to store the data. Next comes the data itself. Each byte is stored as soon as it is received. After the data is a "/" (hex $2F) which indicates the end of data. If this is not received, the SYM will issue an error message. A checksum follows to verify that the data has loaded correctly. After the checksum are two EOT (hex $04) characters to indicate the end of the file.

This is how SUPERMON stores data and is by no means the only method. The only part you should not alter is the series of sync characters.

To create a file of your own format, load 9 into the accumulator and execute subroutine CONFIG in SUPERMON ($89A5). This configures the I/O port to run the tape interface. Put $EC in $A00C to turn on the tape recorder motor. Now initialize your own variables such as checksums, counters and pointers.

Put out a series of sync characters by loading $16 in the accumulator and calling OUTBTH for a few seconds. OUTBTH destroys the accumulator contents so it must be reloaded each time.

A known non-sync character should follow to indicate the end of the sync sequence. If you use a character other than "*", this could indicate the tape format and your program could jump into SUPERMON's cassette routine or your own automatically.

After this it is entirely up to you. Try to include some checksums but remember, the more "garbage" you incude, the slower the system will appear.

To read a file, repeat the initialization sequence required for saving data. Instead of storing sync characters we must now search for them. This is done simply by calling subroutine SYNC after storing $80 in location $FD to indicate that the tape will be in the high speed format. When the sync is found the subroutine will return to your routine.

The next task is to search for the end of sync character. Simply call, RDBYTH and examine the contents of the accumulator. If the character is the sync character, then read another character from tape and try again until you find the end of sync character.

From here on, the sequence of operations depends on your format. It is important to ensure that a character is processed as quickly as possible. You can not stop the tape because you need a little more time before the next character.

As an exercise, try writing a tape directory routine. It should be able to list ID numbers and start and end addresses of all of the files on a tape.

Next Month - SYSTEM RAM ADDRESSING and EXTENDING SUPERMON

For the past few months I have explained a bit about how SUPERMON works, and many other bits and pieces. All of this information is useless if you can not do anything about it. Next month we will bring all of the information presented over the previous months together and start extending SUPERMON in a way that is transparent to the user. Your own commands will appear identical to SUPERMONs commands.

IMPORTANT NOTE: There are 13 SYM owners in KAOS. I have had four questionaires returned. What has happened to the rest of them?!!!! If you have lost yours, ask Rosemary Eyles for another.

Brian Campbell

****

MORE ON FASTER CASSETTES FOR THE SUPERBOARD

The standard cassette port on the Superboard operates at 300 Baud. At this rate it takes around seven minutes to transfer 4K of data. This is not the first, nor likely the last, mod. to be devised to speed up data transfer without resorting to disks.

Once operating, that 4K of data can be transferred in approximately 25 seconds.

If your Superboard is able to run with CPU at 2 MHz then 1200 Baud operation is possible as soon as you've installed the extra bits.

In BASIC, with the CPU running at 2 MHz, 1200 Baud may be used if an extra 8 nulls are inserted after the LF's in the SAVE mode (NULL 8). No additional nulls are necessary when working with either the ASS/ED or the EX.MON.

Next page please

To use 2400 Baud it is necessary to use a machine code program to talk to the cassette port. The CPU need only be operating at 1 MHz in this case. Greg Kilfoyle provided a suitable program in the KAOS newsletter Vol.1 No.9, p 9. I intend to make my program available through KAOS's library.

Switch gang A selects the appropriate frequency to feed to the ACIA to obtain the desired Baud rate.

Gang B switches a resistor combination in parallel with an existing resistor on the Superboard. This modifies the time constant that is used to decode the 1;s and 0's from the tape.

Gang C: The frequencies generated by the cassette port, when operating at 1200 Baud, are four times those generated at 300 Baud i.e. 4800 & 9600 Hz. The LS193 divides these frequencies by two for the 1200 Baud setting, and by four for the 2400 Baud setting.

This means that at 1200 Baud 0's are represented by four cycles of 4800 Hz and 1's by two cycles of 2400 Hz.
Similarly at 2400 Baud, 0's - two cycles of 4800 Hz, 1's - one cycle of 2400 Hz.

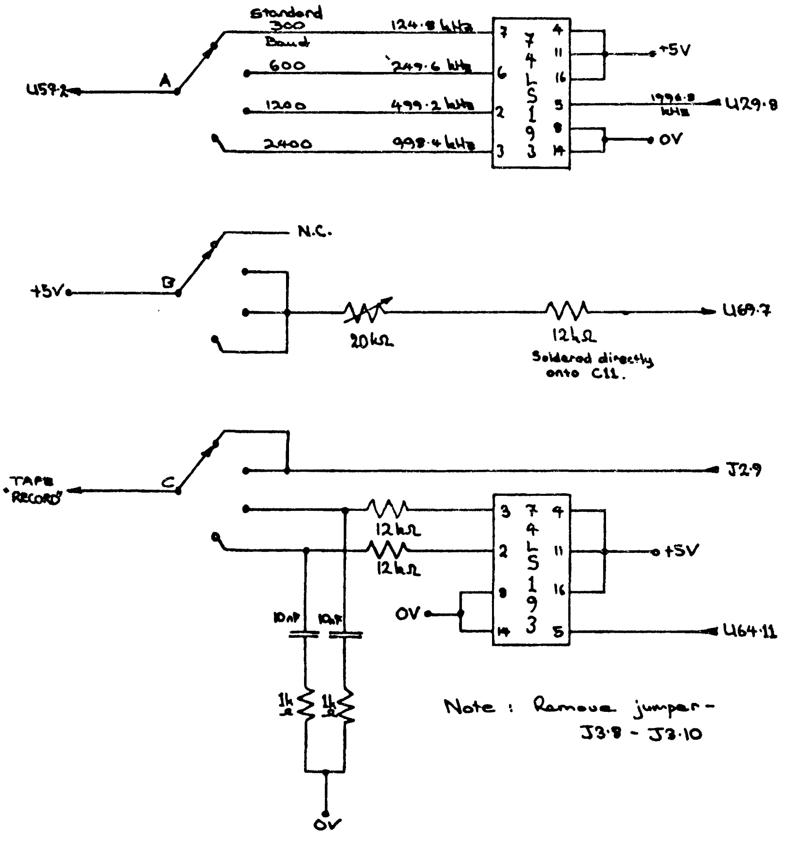
I have been using the 1200 Baud rate for three or four months with 100% success. In my experience at 2400 Baud the success rate drops to about 90%.

I attribute the lost 10% to tape drop outs, consequently if 90% isn't good enough, spend a bit more time and money to hunt up better quality tapes (I'm happy with the club's tapes at $1.00 each).

There are two resistors that can be fiddled with, the 20K trimmer in the mod. and R57 on the Superboard. I have found that adjustment of these two resistors is not critical and suggest initially, that R57 be left alone with the 20K trimmer set midway.

To adjust these resistors more accurately I set R57 midway between the settings at which failure occurred when loading tapes at 300 Baud. Similarly for the trimmer, operating at 1200 Baud.

Peter Careless



Note : Remove jumper - J3.8 - J3.10

Hardware modifications to speed up cassette port.

PRECISION BAUD RATE GENERATOR

One of the problems I experienced when I obtained my C4P was in loading tapes made on my Superboard. A study of the C4P Serial interface showed that the Baud Rate Generator consisted of a 555 timer IC and a simple RC network which is supposed to oscillate at 4800 Hz (4800/16=300). The Superboard obtains its Baud Rate by dividing down the main System Crystal clock to obtain its Baud Rate which is slightly slow but stable.

The C4P RC Network may be adjusted to 4800 Hz but drifts with temp. change, it is also very hard to obtain other Baud Rates at the touch of a switch, thus the following simple mod came about.

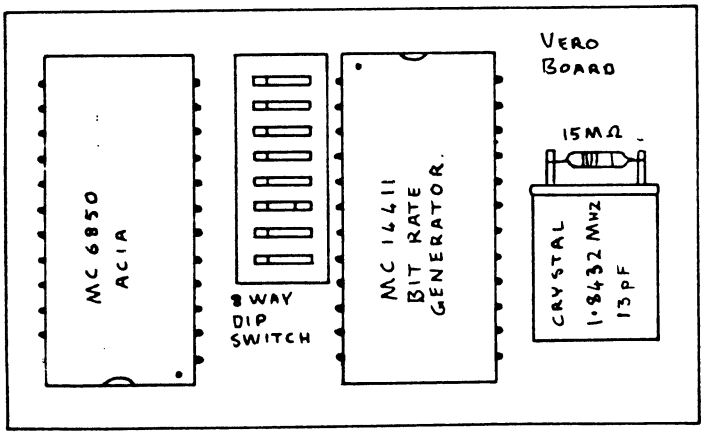
This circuit is based on the Motorola MC 14411 CMOS Bit Rate Generator IC. This IC when used with a 1.8432 MHz Crystal is capable of producing Multiple Baud rates between 614.4 KHz down to 75 Hz. 14 different rates are available at any time on output pins F1 > F14 and by changing the 2 bit binary code on the Rate Select pins A & B to select 1 of 4 internal registers. I have set up this circuit so that RSa is low and RSb is high, this gives us the most useful range when the ACIA is working in its usual divide by 16 mode.

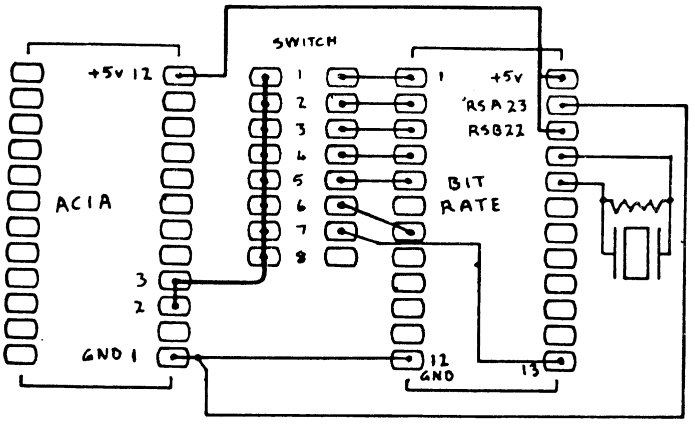
This mod was constructed on a small piece of Vero board just large enough to hold the components. Fitting is pure simplicity, just remove your ACIA, plug in this board and refit your ACIA into the new board. No modifications are required to the computer.

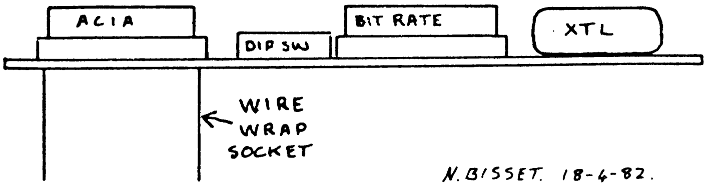
The wire wrap socket that passes through our new board to plug into our computer should have pins 3 and 4 cut off below the new board so that the timing signals from the computer are stopped from reaching the ACIA and are replaced by our new signals, all other ACIA are unaffected. If you require to keep changing the rate then I suggest you leave out the dip switch and use ribbon cable to an external rotary switch.

Nigel Bisset

NOTE Location of Pin 1 on Each IC.







N. BISSET 18-4-82.

SWITCH	BIT RATE	BAUD RATE ÷ 16
1	153600	9600
2	76800	4800
3	38400	2400
4	19200	1200
5	9600	600
6	4800	300
7	1760	110
8	NOT	USED

SUPERBOARD

Hello again from the Ohio Superboard User Group in Queensland. First, a little more about us, followed by more items from past newsletters.

We started in May, 1980, and have over 100 members in Australia & New Zealand. Very few of our members have disk, so most of our articles are about the machine as it comes. We also realise that most of the members are not technical geniuses, so our projects are simple, but useful.

From next month, our regular newsletter will be published in the KAOS journal, but for new members there are 25 back copies available. Each year we run a programming competition. This year, the March Competition attracted 13 entries of all types, for prizemoney totalling $60.00.

With the publication of our newsletter in KAOS, our expenses are reduced dramaticly, and I expect that our membership fee will be our only charge for the lifetime of the club. Address of the Ohio Superboard User Group is :-   146 York Street, Nundah, QLD. 4012,if you would like to join.

MESSAGE PRINTER

Don't let your computer be the boss. Give it some humility with this routine, and impress your friends. POKE 5,168 for OK message.

```
50000 DATA 160,2,169,202,76,195,168
50010 DATA 13,10,82,101,97,100,121,44,77,97,115,116,101,
      114,13,10,0
50020 FORR=707 TO 730:READZ:POKE R,Z:NEXT:POKE 5,2
```

Seriously, a better use is to automatically switch off LOAD and RUN a program. To do this, change the data lines as below.

```
50010 DATA 13,10,80,79,75,69,53,49,53,44,48,58,82,85,78,
      13,10,0
50020 FORR=707 TO 731:READZ:POKE R,Z:NEXT:POKE 5,2
```

OUT OF MEMORY

If you just need another hundred bytes for your program when that OM ERROR comes up, this routine will give it to you.

```
POKE 565,0:POKE 121,54:POKE 122,2:NEW
```

Now if you ?FRE(9), you will find you have another 200 bytes free. Of course you have to SAVE your part finished program to tape, do the change, then LOAD again. Also, you will want to set up the extra memory before loading again. Here's how to get your program to do it!

```
0 PRINT:PRINT"POKE565,0:POKE121,54:POKE122,2:NEW":LIST 1-
```

Enter line 0 as above as part of your program. To save it and your long program to tape, type SAVE:RUN

C1P FAST SCREEN CLEAR

This is nearly as fast as a machine code screen clear. make it a subroutine and call it as often as required.

```
1000 A=PEEK(129):B=PEEK(130):POKE 129,0:POKE 130,212
1010 S$=" ":FOR S=1 TO 7:S$=S$+S$+" ":NEXT
1020 POKE 129,A:POKE 130,B:RETURN
```

SUPERBOARD

RANDOM SELECTION METHODS

How do you set up a computer if you want to draw out a full set of numbers once only, as in " Bingo ". With numbers 1 to 99, try to figure out a good method yourself before you try this routine.

```
10 DIM A(100):FOR R=1 TO 99:A(R)=R:NEXT R
20 FOR R=1 TO 99:X=INT((100-R)*RND(9)+1)
30 PRINT A(X);:A(X)=A(100-R):NEXT R
```

GOSUB INPUT

This routine replaces INPUT, and allows you to return with any string, even nothing. It won't jump out to command mode, print REDO or other such messages. Input any character including " ; , without the EXTRA IGNORED message. The one exception is @, which aborts all previous input, but otherwise causes no problems.

```
 999 PRINT"INPUT ANY STRING";:GOSUB 1000 :PRINT S$:END
1000 POKE 11,87:POKE 12,163:X=USR(X):AD=19
1010 IF PEEK(AD)=0 THEN RETURN
1020 S$=S$+CHR$(PEEK(AD)):AD=AD+1:GOTO 1010
```

DETECT CONTROL KEYS

Values for right and left shift, repeat, control and escape keys are continuously available at location 57100. These are very useful for games because you don't need to poke to the keyboard row. There is also no need to turn off the control C check routine. The keys are also easier to use in M/C games, the shift keys making excellent left and right movement controls and repeat and control being good for firing.

With shift lock down, the values are:-

NO KEY PRESSED	254	REPEAT	126
RIGHT SHIFT	252	CONTROL	255
LEFT SHIFT	250	ESCAPE	222

DEFEAT THOSE PROGRAM EXITS

Over the last 24 issues of the newsletter, many uses have been found for the OK message printer at locations 0004 and 0005. This one makes sure that a friend playing a game on your machine cannot cause a jump out to command mode by pressing the return key without an input. This won't prevent this from happening if the break key is pressed, and worse, a warmstart won't work.

Use these lines in your program:-

```
10 DATA 32,101,166,104,104,76,252,165
20 FOR R=707 TO 714:READ Z:POKE R,Z:NEXT:POKE 5,2
```

To restore the warmstart, end your program with POKE 5,168:END

See you next month,
Ed.

IMPLEMENTING MODIFICATIONS TO DISK BASIC

I have postponed my description of a CALL statement for BASIC and I will try and describe how to implement the changes to BASIC given last month. The general method is to load the relevant track(s) to memory, modify the relevant bytes and then write the modified track back to the disk. The modifications should not be implemented 'on the fly' (by POKEing into BASIC) for two reasons.

Firstly, some parts of BASIC are self-modifying, so that the code as it executes may not be the same as the code on the disk. Secondly, if BASIC tries to execute a partially complete patch, it is almost guaranteed to crash. Hence it is normal to load the track into a position different from that where it usually resides. The modifications could probably be done ith the Assembler, Extended Monitor or (if you are a masochist) even using Disk Basic (converting all the hex to decimal). It is easiest to use the Extended Monitor.

The modifications described last month pertain to track 4 (on 5¼" disks) which belongs in 1200 to 19FF, thus 17A5 (the first byte of the corrections) is offset 05A5 (965 decimal) bytes from the start of the track.
The modifications could be made as follows: (Your commands underlined)

```
A*EM (CR)                 go into the Extended Monitor
:!CA 4200=04,1 (CR)       read the track into the workspace
:@47A5 (CR)               open the byte you wish to change
47A5/90 90 (LF)           type in the correction
47A6/40 0E (CR)
:@47AB (CR)               open the location for the next correction
47AB/A9 66 (LF)
47AC/00 AF (LF)           etc.
```

When all the corrections are complete, use the Q command to check them:

```
:Q47A5(CR)
```

and if everything is okay then save the new version of track 4.

```
:!SA 04,1=4200/8(CR)
```

Check that BASIC still works, especially that arithmetic operations still work. Try PRINT SQR(10). If the answer is not 3.16227766 then the correction did not work. In this case try comparing your patched track 4 with an unpatched one (don't attempt this sort of mod on your only disk!).

There are three sets of modifications (17A5 to 17B5, 184A to 1862 and 1946 to 1954) that are independant of each other and I would recommend implementing them one at a time to aid debugging.

Note that there was an error in the machine code that I gave last month (all my fault), 1953 should be D6 not 96. The assembler BNE 192A is correct.

If you have 8" disks, the procedure is much the same except that some of the numbers change. The corrections apply to track 3 which belongs (?) in 0E00 to 1A00, so all that need change is the CALL/SAVE commands:

```
:!CA 3E00=03,1 (CR)  and
:!SA 03,1=3E00/C (CR)
```

The procedure is the same to correct the OPEN command, except that 2F18 resides on track 12, sector 4 (except on 8" disks, where it is track 8) so the commands required are:

```
:!CA 3E79=12,4 (CR)
:@3F18 (CR)
3F18/AD 20 (LF)
3F19/92 6F (LF)
3F1A/2F 19 (CR)
:!SA 12,4=3E79/1 (CR)
```

Then insert the subroutine into the space you have made on track 4, using the same techniques again. You should be an expert with the Extended Monitor by now!
Hope this makes it easier for you to get the 'fast' Disk Basic working.

Rodney Eisfelder

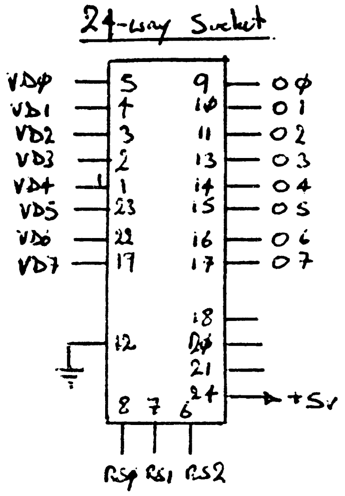
****

PROGRAMMABLE CHARACTERS

(for the TASAN Video Board)

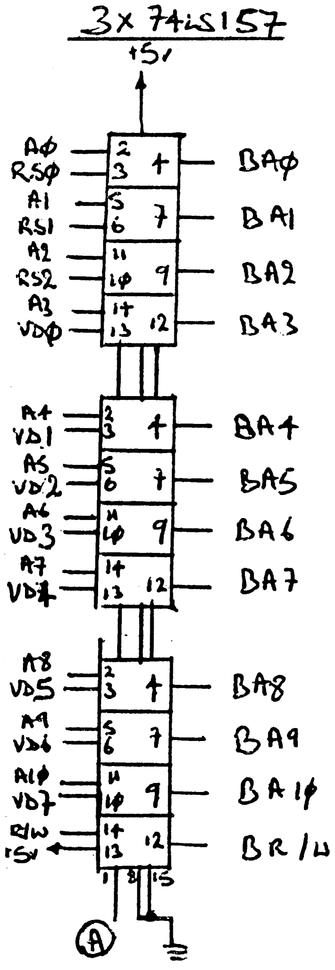
This circuit should be constructed on a piece of Veroboard, and plugged into the character generator socket of the Video board via a 24 way wirewrap socket. Additional lines from the Video board (address, data busses etc.) could be soldered directly to the Video board and terminated in 16 way headers which plug into the Veroboard.

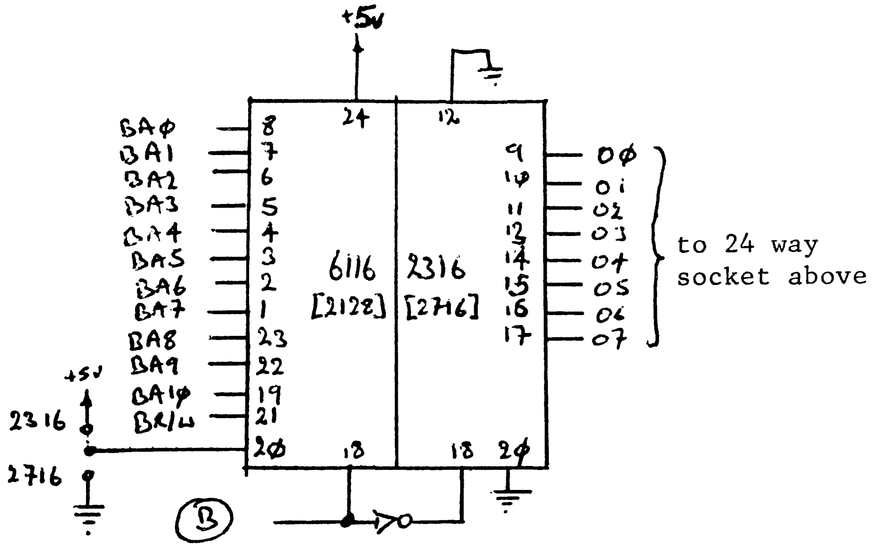
Paul Dodd



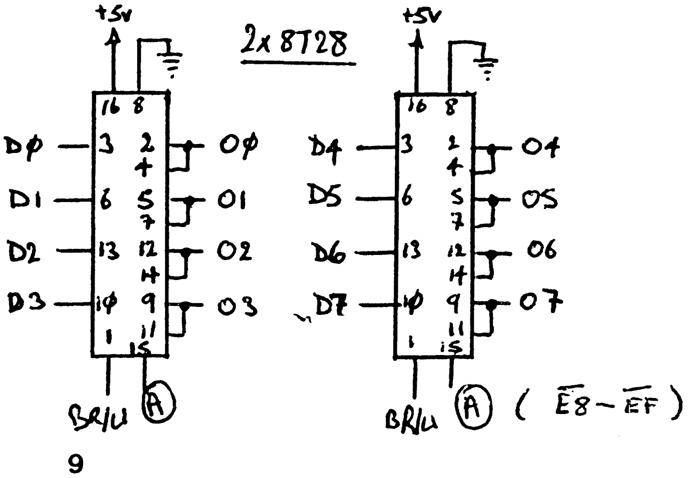
Wire the 2316 (2716) character generator ROM and the 2Kx8 RAM (6116,2128 etc.) together, pin for pin except for pins 18 ($\overline{CS}$) and 20($\overline{OE}$ on RAM,$\overline{CS}$ on ROM)

On Circuit A goes to pin 10 25 on Video board
B goes to pin 15 22 on Video board





to 24 way socket above



MORE OF THE OSI ASSEMBLER REVEALED

Back on page 7 of the November 1981 issue of the KAOS magazine, George Nikolaidis announced the discovery of a 16 bit psuedo machine living in the heart of the OSI Assembler/Editor. Being something of a masochist I decided to investigate.........

Since then I have made a few discoveries and figure that it is time to pass on what I know to anyone else who happens to be interested in this sort of thing.

The key to the assemblers operation is an interpreter routine located at $1179-$11E8. Once you understand what this piece of code is doing, a few things start to fall into place. The interpreter does the following:

get opcode, prepare for validation
check for a valid opcode ($00-27 or $80-A7)
set up a jump vector pointing to the code which processes that opcode
find the N°of bytes in that opcode and store it for later use in updating the programme counter
jump to the processing routine.

Some of the processing routines are written in the 16 bit code and a few of these are also hybrid. (ie they finish up back in 6502 code) When processing is complete the routines usually go back to the interpreter to update the programme counter and get the next instruction.

A quick comparison of the validation routine and the jump table shows that there are 39 opcodes. Some of these have the high bit on (ie they have values greater than $7F). These are macro instructions (multiple operation) and usually written in the psuedo language.

Using this knowledge, it is now possible to do a complete disassembly of the opcodes in the assembler. I have done this already and have a very rough hand written copy which I work from.

The table shows the actual opcodes used in the assembler together with what little I know of the function of each instruction. Upper case mnemonics courtesy of George; others are my suggestion.

As you can see there is still much to discover. What is needed now is for some more of you people to get to work so we can hurry this up a bit. (This may not look like much, but it's taken hundreds of hours to get this far.)

In documenting this I have deliberately avoided details so that the reader would not be bored with technicality. If you would like to know more, have any ideas on the function of any of the opcodes or would be prepared to help in documenting the assembler, I can be contacted on

David Dodds

Next page please

EXTENDED MONITOR IN 48X12 FORMAT ON A C1P WITH DABUG III

Enter the standard monitor and change 021A to 00 and 021B to FB. Run the Extended Monitor and change 00FB to 01 and 099D to 0B using the @ command.

John Whitehead

SUMMARY OF ASSEMBLER/EDITOR OPCODES

OPCODE (hex)	PROCESSED at $ locn	MNEMONIC	BYTES	FUNCTION
00	135A	INP$	1	Input till c/r
01	0240	CIC A;nn	3	Indexed compare with A;branch $nn if result equal
02	0244	CPC A;nn	3	" " " " " "
03	0270	MOV A,B	3	Move contents of A to B (16 bits)
04	0279	GO nnnn	3	Jump to location $nnnn
05	0284	BRA nn	2	Branch relative to p.c. $nn
06	0363	err	3	Print E# messages
07	037D	txt	*	Output following text till $00 or $0D is found
08	03EC	msg	*	similar to $07
09	03A0		5	byte3 is mask;byte5 is rel. branch
0A	0400	ADD A,B:C	4	16 bit add loc A to B:result in C
0B	0419	SUB A,B:C	4	16 bit substract loc B from Loc A result into C
0C	0440		1	
8D	047C		2	
8E	04C0		4	
8F	03D4		1	
10	0300	GOV	3	byte3 is relative branch
11	02BC	rtm	1	return from macro processing
12	02B8	mbr n	2	Take macro rel. branch N° n
13	02D0	INC A	2	Increment location A (16 bit)
14	02D6	DEC A	2	Decrement " " "
15	02E4	MAC	1	Revert to 6502 code next byte
96	0626		4	last 3 bytes are rel. branch calls
97	02CB		1	Transfer contents of $04 to $00 (16 bit)
18	0965		2	
19	0509		2	
9A	0700		2	
1B	0500		2	
9C	095D		1	
9D	0989		5	
9E	0985		5	
1F	024E	NOP	1	No operation
A0	0C03		2	
A1	0AF0		2	
22	04AF	hmov a,b	3	Move a byte from a to b
23	0A60		1	
24	0380	ASC	*	output a line of text
25	0BE5	HEX	2	does a binary to hex conversion and prints the result
26	0BC5		3	
A7	0E25		1	

* these appear in the table of byte values as 15 bytes. The routines however do their own byte count.

```
1179    A000       LDY #$00        reset to point to first character
117B    4C2C03     JMP $032C       go get opcode and prepare to validate
117E    C94F       CMP #$4F        is it a legal opcode ($00-27 or $80-A7)
1180    9005       BCC $1187       yes branch and continue processing
1182    A624       LDX $24         load prog. count. low
1184    A425       LDY $25           "     "     "    high
1186    00         BRK             stop
1187    18         CLC
1188    6980       ADC #$80        add acc. to base address $0780 to obtain
```

```
118A   85E4     STA $E4        the address of the routine to process that
118C   A900     LDA #$00       opcode
118E   6907     ADC #$07       store result in $E4/E5
1190   85E5     STA $E5
1192   B124     LDA ($24),Y    reget opcode
1194   297F     AND #$7F       mask off high bit
1196   4A       LSR A          divide by 2; bit 0 to carry
1197   AA       TAX            set pointer to byte table
1198   BDEA11   LDA $11EA,X    get 2 byte values
119B   B004     BCS $11A1      was low bit a 1? branch if yes
119D   4A       LSR A
119E   4A       LSR A
119F   4A       LSR A
11A0   4A       LSR A          shift 4msb to lsb
11A1   290F     AND #$0F       mask off 4msb bits
11A3   852A     STA $2A        save byte value
11A5   B124     LDA ($24),Y    reget opcode
11A7   300E     BMI $11B7      if high bit on branch to macro handler
11A9   B1E4     LDA ($E4),Y    get address of processing routine
11AB   85E6     STA $E6        set up jump vector
11AD   C9       INY
11AE   B1E4     LDA ($E4),Y
11B0   85E7     STA $E7
11B2   A200     LDX #$00       reset X reg.
11B4   6CE600   JMP (00E6)     jump to processing routine
11B7   A52A     LDA $2A        get byte count
11B9   48       PHA
11BA   A525     LDA $25        get prog. count. low
11BC   48       PHA
11BD   85FB     STA $FB        set up temp prog counter
11BF   A524     LDA $24        get prog. count. high
11C1   85FA     STA $FA
11C3   48       PHA
11C4   B1E4     LDA ($E4),Y    get new value for prog counter
11C6   8524     STA $24
11C8   C8       INY
11C9   B1E4     LDA ($E4),Y
11CB   8525     STA $25
11CD   D0AA     BNE $1179      branch always get new opcode

032C   A51A     LDA $1A        get source pointer low
032E   8DFE12   STA $12FE      save it
0331   A51B     LDA $1B        get source pointer high
0333   8DFF12   STA $12FF      save it
0336   B124     LDA ($24),Y    get next opcode
0338   0A       ASL A          multiply by 2
0339   4C7E11   JMP $117E      go do validation
```

****

THE MEETING WAS KAOS

Due to the increased number of new members attending meetings, the committee has appointed six people to introduce them to KAOS. In the past, new members have in some cases, been ignored unintentionally. We hope this will solve any future problems and we apologise to those members who have felt like shags on a rock.

AREA MEETINGS:- Because of the number of people attending the monthly meeting, and the difficulty of finding somewhere to have a quiet discussion about things that interest you, the meeting discussed the idea of individual members arranging small gatherings in their own area. These gatherings are not intended to replace the monthly meetings and will not be run by KAOS, it is up to members to make their own arrangements..

RAM BOARD:- Nigel Bisset from Sydney displayed his new 48k static ram board using 6116s. Nigel indicated that the board only draws two tenths of an amp and could also be interfaced with the KIM, SYM and the VIC-20. A C1 to 48 bus adapter is also available to further increase the versatility of the expansion board. The bare board should retail for around $60 and fully constructed boards will also be available soon, the price has not been decided yet..

OSI EXSPANION:- Bill Chilcott hopes to have bare boards available at the June meeting for approximately $85.

POWER SUPPLY:- Compact modular switch mode power supplies were displayed by Sergio Krastic during the meeting. These supplies, which are exstremely efficient, compact and fully protected, can be supplied preset to almost any voltage required. The price, for KAOS members only, is $187 plus tax and they may be purchased from Sergio.

APOLOGIES:- The Eprom Programmer driver routine written by John Whitehead was not based on D.Anears programmer but Tony Van Bergen, which appeared in an earlier KAOS newsletter. Sorry John and Tony.

HI-RES:- A considerable amount of interest was shown in Nigel Bisset's High Resolution video board which was designed by David Anear. Nigel made use of his 48k static ram board to demonstrate the applications of Hi-Res graphics.

P.S.G:- Ron Kerry indicated, during the meeting, that bare boards for his programmable sound generator should be available for around $10 in the near future. The board was designed to be used with the TASKER bus system.

ROD DRYSDALE
VK3BYU

****

DEAR ~~ABBY~~ PAUL

Since there are an ever increasing number of new computer users, all with various problems and questions to ask, we have decided to run a problem page each month. I will try to answer any computer related questions each month in this column. (Note, you do not have to supply your name if you would rather remain anonymous). Anyway, for this month's questions......

Q. Why won't the Dick Smith acoustic coupler work with the TAB?
A. The output frequencies of the DS modem are not crystal locked so they can vary by as much as 10-20% which is just too great a variation for the professional quality modems used by the TAB computer.
Q. What is a data seperator, and why do I need one with my disk drive?
A. A data seperator seperates the data from the clock pulses stored on the disk surface because the disk interface requires seperate lines for the clock and data pulses.
Q. Why do I need to reverse the diodes on my Superboard when I use the new Video board?
A. If you want to run standard C4 software you will need to have a C4 keyboard which is the inverse of the C1's. Hopefully, soon this will not need to be done because at least three KAOS members are working on modifications to give you either a C1 keyboard or a C4 keyboard, fully switchable. This will mean that you can run all standard C4 software, all standard C1 software and a large variety of hybrid software - i.e. C1 software written for a 64 X 32 screen etc.

Cheerio until next month, and keep those questions rolling in!

Paul Dodd

INTERFACING THE APPLE AND OHIO

A cheap way of adding multiple terminals to your school's APPLE II

This system has been devised to work with the APPLE 'High Speed Serial Interface Card' but any of the other RS232 serial cards for the APPLE should do the job. The only modification needed on the APPLE card was a wire link from the -12V rail to a spare line on the connector. This was necessary because the APPLE would not accept the 0V to +5V signal from the C1P. (See fig.1)
The serial card is set to 1200 Baud.
The only modifications that are required on the C1P are:-

1. The baud rate on the Modem position was increased to. 1200 baud instead of 300 baud. 300 baud is the standard Modem baud rate which is too slow for computer to computer communication.
2. The negative 12V line on the connector provided by the interface (see above) must be attached to pin 7 of J3 on the back of the Superboard. Link option W10 which is currently hooked up to ground must be cut, otherwise the negative 12V will be shorted to ground.

*CAUTION Before powering up either system, perform continuity tests to ensure that the above has been performed correctly.

The software to drive the system is quite simple with one program, A2C2 (see listing 1) to send data from the APPLE to the C1 and another, C2A2 (listing 2) for the other direction.

A2C2 exists in the form of a text file on the APPLE disk. After loading the program to be sent to the OHIO, EXEC A2C2 loads this program (lines 0 - 4). When RUN is typed the user is prompted to type 'LOAD' on the C1 and then type 'GO' on the APPLE.

C2A2 is loaded into the C1 and the program to be sent to the APPLE must be contained in lines between 10 and 59998. After RUN the user is prompted to type 'IN#2' on the APPLE (card in slot 2) and 'GO' on the C1. The 3-way switch on the C1 must be set to modem. Multiple C1's can be connected through a simple switching device and any number of these can receive data simultaneously but obviously only one at a time can send.

I have 4 C1's talking to one APPLE at Preshil and have been running the system without any major hassles for over two terms now. The only limitation seems to be that C1 program lines need to be kept reasonably short in order for the APPLE Basic Interpreter to digest properly.

George and I are working on an improved version of the software in order to provide a more flexible and 'friendly' system and will publish the fruits of our labour in a later issue.

Noel Dollman

APPLE TO CHALLENGER LISTING 1 A2C2

```
0  HOME:VTAB8:HTAB6:PRINT"TYPE";:INVERSE:PRINT"LOAD"''"NORMAL:
   PRINT"(CR) ON CHALLENGER":PRINT
1  HTAB6:PRINT"THEN TYPE";:INVERSE:PRINT"GO";:NORMAL:PRINT
   "(CR) ON APPLE":PRINT:PRINT
2  HTAB6:INPUTR$:IFR$< >CHR$(71)+CHR$(79)THEN2
3  POKE33,33:SPEED=150:PR#2:FORX=1TO1000:NEXT:LIST7,
4  PR#0:SPEED=255:TEXT:HOME:PRINTCHR$(7):END
```

CHALLENGER TO APPLE   LISTING 2 C2A2

```
8 REM ************************
9 GOTO 60000
10 REM ************************
59999 END
60000 PRINTCHR$(127):PRINT:PRINT
60010 PRINT"Type  IN#2  on Apple":PRINT"Then type GO on Challenger"
60020 INPUTG$:IFG$="GO"THEN60040
60030 GOTO60020
60040 M=546:FORX=0TO46:READA:POKEM+X,A:NEXT
60050 POKE538,34:POKE539,2
60060 LIST10-59997
60070 END
60080 DATA32,41,255,141,254,2,72,152,72,138,72,173,254,2,201
60090 DATA10,208,20,162,3,134,112,162,255,160,255,136,208,253
60100 DATA202,208,248,198,112,208,242,240,3,32,177,252,104,170
60110 DATA104,168,104,96
```

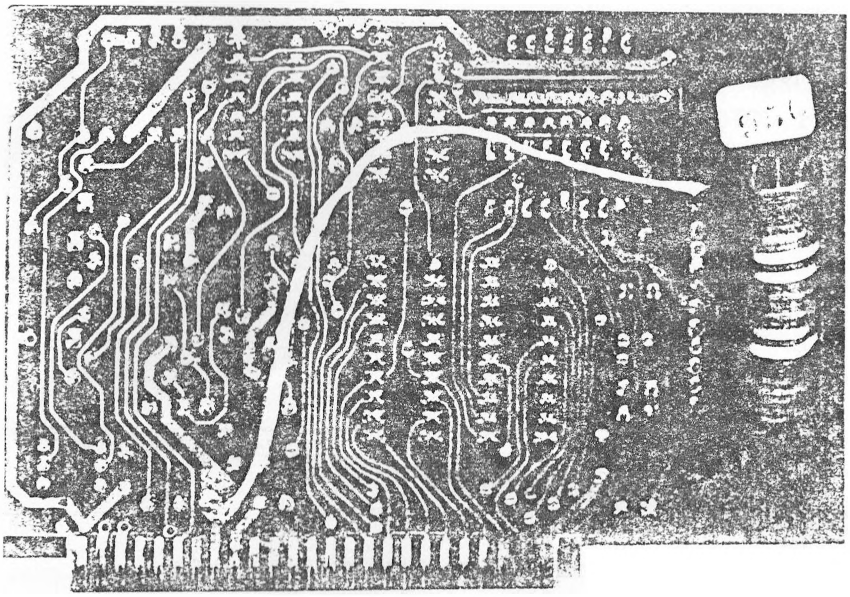


Fig. 1

Back of Apple serial card showing mod.

****

FORTH ON CASSETTE

One of our members, Ritchie Laird, was one of the first to receive a copy of Greg Kilfoyle's cassette version of FORTH and he appears to be very enthusiastic about it, so much so that the last we heard, he was pulling the BASIC ROM out of his Superboard and converting it to a FORTH machine.

Ritchie has versions of FORTH to suit 48 and 64 screen formats and has written a FORTH 6502 Assembler. He is offering these to club members at the cost of cassette and postage or free if cassette and package are supplied.

Ritchie would like any members who are interested in FORTH to contact him with a view to exchanging information and is willing to help any beginners to the language. He is also interested in forming a FORTH Users group. If you are interested, contact Ritchie at

P.S. Because of the cost of postage, it would be appreciated by Ritchie if you enclosed a stamped, addressed envelope when writing to him.

MORE ON DATES

Malcolm Coghill (KAOS N/L Vol.2 No.8 p15-16) gives useful date conversion routines. Further routines and routines for date checking and formatting are given in Microcomputing May 1980, p176-7, 208-212.

When file space is at a premium, the following compression of date may be useful:-

DD = day, MM = month, YY = year, DT$ = date in form   YYMMDD

```
DT$ = CHR$ (YY) + CHR$ (MM+96) + CHR$ (DD+60)
DT$=CHR$(YY)+CHR$(MM+96)+CHR$(DD+60)
YY=VAL(LEFT$(DT$,1)):MM=VAL(MID$(DT$,Z,1))-96:DD=VAL(RIGHT$(DT$,1))-60
```

The following sub-routines for date input, storage and decoding have been adapted from OSI - DBMS programs on 65U.

```
 999 REM    Date Input, Check, Storage
1000 INPUT "Today's Date  (DD/MM/YY)  ";DT$;IF DT$=" "THEN 1000
1010 DD$=" ":MM$=" ":YY$=" ":DX=0
1020 FORJ=1TOLEN(DT$)
1030 DZ$=MID$(DT$,J,1):IF DT$=","OR DZ$="-"THEN DZ$="/"
1040 IF DZ$="/"THEN DX=DX+1:GOTO 1100
1050 IF DZ$ < "0"OR DZ$ > "9"THEN 1000
1060 ON DX+1 GOTO
1070 DD$=DD$+DZ$:GOTO 1100
1080 MM$=MM$+DZ$:GOTO 1100
1090 YY$=YY$+DZ$
1100 IF VAL(DD$)< 1OR VAL(DD$)>31 OR VAL(MM$)< 1 OR VAL(MM$) >12 THEN 1000
1120 IF VAL(YY$)< 82 OR VAL(YY$)  85 THEN 1000
1130 IF(VAL(YY$))/4-4*INT((VAL(YY$))/4)=0 AND VAL(MM$)=2 THEN 1150
1140 GOTO 1170
1150 IF VAL(DD$)>29 THEN 1000
1160 GOTO 1180
1170 IF VAL(DD$) >28 THEN 1000
1180 IF VAL(DD$)<31 THEN 1200
1190 IF VAL(MM$)=4 OR VAL(MM$)=6 OR VAL(MM$)=9 OR VAL(MM$)=11 THEN 1000
1200 P=24569:POKEP,VAL(DD$):POKEP+1,VAL(MM$):POKEP+2,VAL(YY$)
1210 RETURN
1299 REM   RETRIEVE, DECODE DATE
1300 A=24569:DT$=RIGHT$(STR$(PEEK(A)+100,2)+"/"
1310 DT$=RIGHT$(STR$(PEEK(A+1)+100,2)+"/"
1320 DT$=DT$+RIGHT$(STR$(PEEK(A+2)+100,2)
1330 RETURN
```

Frank Nicholls

****